NASA/CR–2014-218504

Distributed System Design Checklist

Brendan Hall and Kevin Driscoll
Honeywell International, Inc., Golden Valley, Minnesota

July 2014

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NASA Aeronautics and Space Database and its public interface, the NASA Technical Report Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at *http://www.sti.nasa.gov*

- E-mail your question to help@sti.nasa.gov

- Fax your question to the NASA STI Information Desk at 443-757-5803

- Phone the NASA STI Information Desk at 443-757-5802

- Write to:
  STI Information Desk
  NASA Center for AeroSpace Information
  7115 Standard Drive
  Hanover, MD 21076-1320

NASA/CR–2014-218504

Distributed System Design Checklist

Brendan Hall and Kevin Driscoll
Honeywell International, Inc., Golden Valley, Minnesota

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NNL10AB32T

July 2014

Acknowledgments

We would like to thank all those who contributed ideas to this checklist, including: Sean M. Beatty, Devesh Bhatt, Ted Bonk, Les Chambers, James S. Gross, Philip Koopman, Carl E. Landwehr, Frank Ortmeier, William T. Smithgall, Matthew Squair, Wilfried Steiner, Ying Chin (Bob) Yeh, other members of the System Safety e-mail list (systemsafety@techfak.uni-bielefeld.de), IFIP Working Group 10.4 (http://www.dependability.org/wg10.4), and employees of Honeywell.

Available from:

NASA Center for AeroSpace Information
7115 Standard Drive
Hanover, MD 21076-1320
443-757-5802

This report describes a design checklist targeted to fault-tolerant distributed electronic systems. Many of the questions and discussions in this checklist may be generally applicable to the development of any safety-critical system. However, the primary focus of this report covers the issues relating to distributed electronic system design.

The questions that comprise this design checklist were created with the intent to stimulate system designers' thought processes in a way that hopefully helps them to establish a broader perspective from which they can assess the system's dependability and fault-tolerance mechanisms. While best effort was expended to make this checklist as comprehensive as possible, it is not (and cannot be) complete. Instead, we expect that this list of questions and the associated rationale for the questions will continue to evolve as lessons are learned and further knowledge is established. In this regard, it is our intent to post the questions of this checklist on a suitable public web-forum, such as the NASA DASH*link* AFCS repository (https://c3.nasa.gov/dashlink/projects/79). From there, we hope that it can be updated, extended, and maintained after our initial research has been completed.

Contents

1 Introduction

This report presents a design checklist to be applied to fault-tolerant, distributed, safety-relevant, software-intensive systems. It has been generated under NASA Task Order NNL10AB32T, Validation and Verification of Safety-Critical Integrated Distributed Systems – Area 2.

1.1 Scope

This report describes a design checklist targeted to fault-tolerant, distributed, safety-relevant, software-intensive systems. Many of the questions and much of the discussion in this checklist may be generally applicable to the development of any safety-critical system.

The questions that comprise this design checklist were created with the intent to stimulate system designers to seeing a broader perspective from which to assess system dependability and fault-tolerance mechanisms. While the checklist is quite comprehensive, it is not (and cannot be) complete. We expect that the questions and their rationales will continue to evolve as lessons are learned and further knowledge gained. We plan to post the checklist questions on a suitable public web-forum, such as the NASA DASH*link* AFCS repository [1], and expect that it will be updated and maintained after our initial research is complete.

1.2 Background and Motivation

Modern avionic systems continue to increase in size and complexity. State-of-the-art flight control systems can consist of more than a million lines of code. The Lockheed Martin F22 Raptor contains approximately 1.7 million lines of code [2] and the next-generation F35 fighter is estimated to comprise 5.7M lines of code. This trend is not limited to the military arena; the software content of the Boeing 787 is estimated at 13 million lines of code [3].

Advancements in networking technology continue to enable increasingly distributed systems and greater interdependence among subsystems that originally used a loosely-coupled, federated design. Network-centric integrated modular avionics (IMA) architectures are the industry norm across all aircraft segments, from large air transport, such as the A380 [4] to general aviation [5]. This integration of multiple aircraft functions into IMA architectures offers many benefits. Standardization on shared hardware platforms may support improved optimization for addressing system obsolescence issues, while simultaneously reducing SWaP (size weight and power).

Similarly, the standardization of software platforms and standard application programming interfaces (APIs), such as ARINC 653 [6] can greatly improve design and system portability, and inter-application communication. Such standardization may enable closer integration and support better levels of cooperation among the traditionally federated aircraft functions. And, as is argued by Rushby [7], this trend may facilitate greater levels of system safety.

However, the increased functionality and higher-levels of integration also change the nature of integrated systems. Aircraft computation architectures are evolving into distributed system-of-systems architectures. This evolution may increase system risks in the form of common-mode or coupled influences originating from the networking platform or being propagated by it, respectively. The latter coupling includes the interaction between the state-spaces of the functions hosted by the system.

This situation is complicated further by the evolutionary nature of typical aircraft system hardware components and architectures. Clean-sheet designs are infrequent. It is more common for the industry to adapt and evolve legacy hardware platforms and systems. Similarly, with the high cost-of-change for software, there is a strong need to re-use software functionality, potentially

porting software systems to different hardware platforms. Therefore, it is important to ensure that the assumptions that underpin the original software and hardware system designs are clearly documented. Additionally, it is important to ensure that these assumptions remain valid as the software and hardware systems are evolved through their respective life-cycles.

The Future Airborne Capability Environment (FACE) [8] is a recent, consortium-led effort addressing some of these challenges. The emerging standard appears to be mainly focused on software (and API issues in particular), and the system issues relating to fault-tolerance or distributed system design are not elaborated in detail.

Similarly, although the certification guidance documentation [9, 10] provides useful criteria and information to guide the design and development process of aircraft functions, it does not give details for assessing the soundness of a fault-tolerant distributed architecture. This lack is understandable, as adding such prescriptive detail within higher-level standards may render the guidance impractical or unworkable. Such lower-level design guidance is better covered by targeted component design guidelines such as in DO-178C [11] and DO-254 [12]. However, these component assurance guidelines also do not incorporate specific details or measures to assess the soundness of a distributed, fault-tolerant system architecture.

DO-297 [10] introduces many areas of system consideration. These areas of consideration are too high-level to address more subtle issues, such as those covered in our parallel research sponsored by the FAA [13]. In that work, we examined the assumptions that underpin the use of cyclic redundancy (CRC) and checksum codes within aerospace systems. Such codes are used in almost all aerospace systems and many ARINC standards such as ARINC 825 [14] and ARINC 664 [15] mandate the use of specific codes. Guidance in relation to how the effectiveness of a given CRC or checksum code should be evaluated does not exist. This problem is not specific to the aerospace industry. For example, the Japan Automotive Software Platform and Architecures - Functional Safety working group [16] recommends the use of a CRC but does not specify the methods to select and assess the suitability of any particular CRC.

The lack of guidelines for using CRCs is of particular interest within distributed systems that typically leverage such codes towards communication system integrity claims. Without path replication or higher-level voting strategies, the integrity of a system is often characterized by the properties of the selected CRC in conjunction with the assumed fault-model of the underlying communication channel (including bit error ratios, burst lengths, inter-symbol interference, etc.). In our FAA report [13], we discuss these issues in detail. From our work in this area, we believe that more prescriptive guidance may be beneficial.

Finally, from our experiences working with the designers of fault-tolerant systems, we find it useful for engineers to examine the validity of their assumptions[1] within the context of the larger system. This simple process can be very effective in removing system design errors or omissions. For example, during a recent discussion about the assumptions that underpinned a replicated task set for a command-monitor architecture, we found omissions within the logic for establishing and maintaining agreement throughout all system startup and fault scenarios. These types of edge-cases are often not discovered during system operation and test[2] and vulnerabilities may manifest as latent design errors if not systematically removed. In this case, the system had been patched for discrete instances where the agreement failed, but the patches did not address the underlying issues that led to agreement failure.

[1] We often ask such questions as we work to establish formal and informal working models of systems that we troubleshoot.

[2] This is another motivation for establishing an architectural-level metric to assess the effectiveness and coverage of system verification and validation activities with respect to system-level fault-tolerance protocols and strategies.

From the above discussion, we believe that assembling a checklist for dependable distributed-system architectures will be beneficial.

2 Research Overview

2.1 Methodology

At the outset of this research, we recognized that developing a useful checklist for distributed fault-tolerant systems was not a small undertaking. We recognized that a suitable checklist must assimilate and integrate knowledge from a wide range of system perspectives, so we developed a multi-pronged approach for generating the checklist. This approach allowed us to solicit input from a number of different areas, sources, and groups as described below:

- We collected, reviewed, and examined existing design-related checklists in use within Honeywell. Honeywell has been a pioneer in integrated avionics architectures, and a tremendous body of knowledge has been captured within the working checklists and system review criteria.
  Note: To this end, we used the Honeywell Goldfire database, a tool for organization knowledge management. The initial query identified over 100 relevant documents in use throughout the organization.

- We then polled experienced senior personnel within Honeywell including many chief engineers and system engineers who have developed distributed and safety-relevant systems.

- Next, we reviewed documented system failures collected under a separate task of this research (posted to the web [17]) to uncover themes and vulnerabilities.

- Finally, to ensure that our checklist did not become too Honeywell-centric, we solicited input from the System Safety Mailing List [18] and the International Foundation for Information Processing (IFIP) Working Group 10.4 [19]. The feedback that we received from these groups is greatly appreciated.

When polling these groups for items that we should include in this checklist, we posed these questions:

- Meta-question:
  "If you were asked to participate in a design review of a safety-critical design, what questions would you ask?"

- Reverse meta-question:
  "If you were presenting a design, what questions would you dread being asked?"

2.2 A Word of Caution about Checklists

During the development of our checklist, we polled and contacted several groups. A recurring theme in the responses was the need for caution on checklist usage. Although many of our respondents recognized the value in assembling information in this format, they reminded us that the blind application of checklists, for example where a user checks boxes without a full understanding of the context or rationale behind the box questions, may deplete the working knowledge of system engineering. From our previous research [20], we were very aware of these issues, and in this

checklist we have tried to avoid yes/no answers and instead solicit the corresponding rationale to be evident within the checklist responses. We also believe that relating the questions to real-world system failure scenarios will help establish the context for the questions.

An additional word of caution was given by Dr. Nancy Leveson who emphasized that looking at the design alone is insufficient to ensure system safety; that to address the system safety, it is necessary to start at the higher-level system safety and hazard-related analyses. While we agree with Dr. Leveson's caution, we also feel that the content of our design-centric checklist will be valuable. Many failures that we have observed have resulted from misunderstanding or ignorance of fault-tolerance theory, which can lead to misaligned or incomplete assumptions about component or system failure modes. In these areas, we believe that the application of our checklist will be beneficial. We believe the application of our list can lead to a more informed view of the system fault-tolerance mechanisms, which will in turn assist system designers and safety analysts to achieve a more informed safety/hazard assessment.[3]

2.3 List Organization and Structure

During the checklist development, we had difficulty settling on a final structure for the questions, given that many of the themes were highly inter-related. Developing a good taxonomy is always a difficult task. In addition, one research goal was to link the questions with suitable background information and lessons learned with respect to real-world systems. To this end, we evaluated different technologies and tools such as mind mapping to support the visualization of the relationships between the questions with the experiences from the "Real System Failures" data set [17] correlated under related AFCS research activity [1]. We selected TheBrain [21] from Brain Technologies as our tool of choice for the following reasons.

- It allows complex relationships to be inter-linked and visualized. It is unique in its abilities to cross-link complex relationships via its three dimensional rendering.
- It has a free viewer application so we did not need to purchase a program to view our output.
- It allows attachments to be connected to thoughts, enabling a simple method to cross-link the information from the "Real System Failures" stories.
- Knowledge captured within TheBrain's tool can also be exported to HTML for hosting on a standard web-server. As discussed in section 4, we will use this aspect to publish the questions to the NASA DASH*link* AFCS repository [1].

A second area of debate during the checklist development related to the number and level of questions included. While some argue that a relatively small number of questions with supporting rationale would be sufficient, other team members were concerned that the subtleties underlying some questions may be left out of higher-level questions, and a set of more detailed questions may prove more useful to assist a less experienced reader. We have decided to include more questions to prevent information loss.

[3] We believe significant improvements in system safety engineering may be achieved by integrating the knowledge bases of system safety engineers and system designers.

3 Checklist Questions

This section contains the checklist questions, criteria, and some rationale for the usefulness of each question. The order of the question presentation is not significant.

3.1 System Requirements

Although this checklist is targeted at system design, we include some questions related to system requirements. As discussed previously, for safety-relevant systems it is important to approach the design from an informed assessment of the potential system hazards and system environment. Designers are encouraged to regularly re-read the published recommended practices in these areas [9, 22, 23] and stay familiar with the material released from related working groups [24].

- Who owns the system requirements?
- Who is responsible for creating and maintaining the requirements and integrating them across different functions?
- Describe your process approach to hazard analysis and requirements definition.
- Highlight all hazards based on operational experience and past history of accidents.
  - How do you know that those you are querying have sufficient operational experience. Please see the related questions of section 3.18.
- Have the *safety requirements* been stated unambiguously (i.e., explicit, clear, concise, understandable, only one interpretation by all stakeholders) and completely?
  - How do you know?
  - If you can't be certain, what is your confidence level and how was that confidence derived?
- Have normal and degraded function modes of operation been identified?
- Have prohibited behavior characteristics been explicitly stated?
- Have requirements been stated in quantitative terms with tolerances where applicable?
- What has been done to check requirements for inconsistencies?
- What steps have been taken to show that the requirements are verifiable?
- What have you done to ensure that the prevailing safety requirements include all functional requirements and all non-functional requirements—qualitatively (e.g., integrity/assurance levels) or quantitatively (e.g., maximum occurrence rates)?
- Have you unambiguously and completely specified the range of operating conditions under which the safety requirements must be met?
  - Where are these requirements documented?
  - What have you done to ensure that the list of assumed operating conditions is complete?
  - If you can't be certain, what is your confidence level and how was that confidence derived?

- What is your strategy for demonstrating that all the safety requirements will be satisfied in the design?
  - Do you have scientifically sound evidence that the safety-critical design meets the safety requirements?
    * Has this evidence been examined by an independent expert and validated to be scientifically sound for this purpose?
    * For any elements of your design that are included to mitigate human failure, how have you determined that the failure probability due to the additional system complexity is not worse than the probability and outcome of the human failure?

3.2 Fault-Hypothesis and Failure Mode Assumptions

In safety-relevant systems, an understanding of the system fault-hypothesis is very important. Without a formal understanding of the system fault-tolerance rationale, it is not possible to validate the system claims. However, in many systems, this aspect of the system design is seldom formalized. In place of a formal fault-hypothesis or formal fault-tolerance specification, non-specific terms such as *single-fault tolerant*, *N-fault tolerant* are commonly used. Until the requirements define the types of faults to be tolerated, the requirements are incomplete and unverifiable. In distributed systems, the fault-hypothesis is central to the system specification since the distributed nature of the fault-tolerance logic often places constraints on the assumed fault model, e.g., the number of simultaneously active faults or minimum fault arrival period. Hence, the following questions are intended to examine the understanding and formality of the system fault-hypothesis.

- What is the system fault-hypothesis?
- Does the fault-hypothesis characterize the failure modes in addition to the number of tolerable faults?
- What are the probabilities for each of the failure modes and how are these probabilities determined?
- For each failure mode class that was not included in your fault hypothesis, what is your rationale for not including it? In particular, for each failure mode class in the following list not included in your fault-hypothesis, supply that rationale:
  - Byzantine [25, 26] (asymmetric, slightly-out-of-specification, sliding value [27, p. 16])
  - In-normnal-path (intended signal propagation path) but out-of-band (over/under: voltage, amperage, pressure, timing, etc.)
  - Out-of-normnal-path (not an intended signal or data propagation path) (internal shrapnel, thermal conduction, vibration)
- Does the fault-hypothesis include both permanent and transient [4] faults?
  - How many permanent and transient faults or combination of permanent and transient faults are tolerated by the system?
  - What is the minimum assumed fault arrival rate?
  - How does system fault-diagnosis impact limit the maximum fault-arrival rate?

[4] a useful definition for transient fault is a fault that has failure duration less than system's fault response/repair time

Note that in fault-tolerant distributed systems, it is common to use distributed fault diagnosis and reconfiguration algorithms to mitigate component failures. These algorithms often cannot tolerate multiple faults within the system diagnosis and reconfiguration windows. Hence, such design strategies may implicitly limit the assumed fault arrival rate.

- How does the system fault-tolerance degrade following the arrival of the first fault?
- Are there other system modes, such as power-down or startup, where the system fault-tolerance can be degraded? See section 3.10 for more questions related to system startup.
- How will the system behave when it is outside of its design envelope?
  - Has any analysis or testing validated this assumed behavior?
- If the number of faults ever exceeds the maximum number of faults the system was designed to handle, will the system recover and resume normal operation when the number of faults falls below the tolerated threshold?
  - How long will it take for the system to recover?
  - Does any persistent state, for example fault-diagnosis records, remain following the system recovery?
  - How is this data used?
- Is the system sensitive to a particular fault sequence?
  - How is this order captured within the system safety analysis?
- Are any faults undetected or not obvious when they occur?

Latent faults are those that go undetected when they occur. Part of the Failure Mode and Effects Analysis (FMEA) process is to determine how detectable each postulated failure mode is. If it is determined that a failure goes undetected when it occurs, the analysis should consider what happens when a second fault occurs. For example, if a function monitor fails undetected, the analysis should address the effect of the function failing undetected now that the monitor is inoperative. The monitor will remain inactive until some explicit check is performed to verify its condition. This in situ verification of the monitor function is often referred to as ***monitor scrubbing****. The coverage of the monitor function should be degraded by the interval of this monitor scrubbing period. That is, the probability that the monitor function works correctly must account for the probability that the monitor itself may have suffered a failure since the last time it was scrubbed.*

- When and how are covering and monitoring functions exercised? *The rationale and assumptions that underpin scrubbing functions should be carefully analyzed—especially for guardian type covering functions that are intended to mask faults and errors. Verifying that these masking functions are operational incurs risk associated with the injection of erroneous and potentially dangerous fault stimulus; pay special attention to ensure that the correct level of interlocks exist.*
- If power-on based monitor scrubbing is assumed, what is the evidence that ensures that the operational envelope of the system matches the monitor scrubbing periods assumed by the system safety assessment? Where is this evidence documented?
  - Is the monitor scrubbing period valid for all system use-case scenarios?

  - Where is this documented?
  - What prevents users from modifying the system usage scenarios beyond the assumptions made above?

The questions above are intended to assist systematic consideration of the distributed system fault-tolerance assumptions. However, this consideration may still be subject to linear thinking limitations. The following questions can help system designers to adopt a wider perspective.

- Have all plausible causes of system failure been considered?
- What if part of the system... is destroyed by fire... is exposed to liquid... falls out of the equipment rack... ?
  - What consequences have been considered?
  - Are the consequences that are captured in the system fault model communicated to the interfacing system? See section 3.8.
  - How will aircraft systems respond correctly to deal with the loss?
- Which transmogrification faults (a component turns into a different type of component) have been considered?
  As exemplified by the failing diode of NASA space shuttle mission STS-124 turning into a capacitor [27, pp. 4–9], the relatively simple failure mode of a simple component may seriously impact system redundancy assumptions if the implications of the failure mode are not fully understood.
- Which "partogenesis" faults (an electrical component appears to be created from nothing) have been considered?
  Examples of partogenesis include capacitors being created by large increases in parasitic capacitance [27, p. 35], and unanticipated emergent properties creating an air conditioner [27, p. 36].
- Is the system vulnerable to a malicious failure? See the glossary for the definition of malicious failure.
  - Think about what would happen if a motivated adversary could make malicious, unconstrained changes to any component of the system (including both hardware and software). What is the worst they could do?
  - How do you know that your answer is really true?
  - For systems that are intended to tolerate multiple faults, consider that the adversary is given access to multiple components. Does this scenario require additional constraints to be introduced into the assumed fault-model?

  Note that additional constraints with respect to multiple faults are not uncommon. For example, most self-checking pair architectures assume that each half of the pair does not fail with identical failure modes at the same time. In the BRAIN [28], this assumption is formalized within the non-colluding fault-model. We believe that spending the time to formalize such details can be very beneficial. Any formal analysis of the systems requires such assumptions to be formally defined.

  Another potential issues arises from the traditional split between safety and system engineering roles. Such a split may result in different failure rates and mode assumptions.

- Does the safety analysis include assessments from all the disciplines and domains of expertise applicable to the system? How has any differences of opinion been reconciled? Where is this documented?
- What ensures that the safety analysis does not assume unrealistic failure rates, either too optimistic or too pessimistic?
- Is the safety analysis consistent with respect to the assumed failure modes?

Be careful to ensure that the failure modes account for physical and logical failure modes. For example, for a bus that incorporates an in-line integrity protection code, the analysis of the bus should include the physical failure modes of the bus (e.g., open, short, non-terminated) as well as the logical potential data corruption failures.

3.3 Failure Containment and Hardware Partitioning

Fault containment regions and hardware partitioning strategies are closely related to the system's fault-hypothesis. These strategies constitute the tool box of the fault-tolerant, distributed-system architect. Implementing such strategies can be difficult, but must be done effectively because there can be no fault tolerance without fault containment. The following questions explore this aspect of distributed system design in more detail.

- For each item in a fault tree (or similar analysis) that is assumed to fail independently, what mechanisms exist in the system's design that enforce this independence?
- What fault-containment regions (FCRs) or hardware partitioning strategies have been defined within the system architecture?
- How are the FCRs specified?
  - Does each FCR have a documented fault-model?
  - Where is the list of failures or errors that can escape each FCR documented?

  Ideally, the definition of the FCRs will be expressed in terms of the formal system fault model. It should define which faults and associated errors are mitigated within the FCR and also specify the faults and errors that the FCR does not handle, which are subsequently exported from the FCR.
- Do any fault-containment regions partially overlap?
  - If so, where and why?
  - Is this overlap consistent with the fault-hypothesis?
  - Can a failure propagation chain change failure modes such that the chain escapes multiple fault containment regions?

Power distribution and management may introduce additional fault-propagation paths. Examining the impact of power-distribution related faults is therefore important. Often with power-distribution, striving for increased levels of availability can compromise the system integrity. Similarly, additional integrity protection in the power-system can reduce system availability. The lessons learned from the F16 power distribution [27, p. 25] may be useful background.

- Can any single point failures disable separate or redundant power supplies? The design should consider the potential of a single point or likely combination of events that can incapacitate redundant power supplies. Steps should be taken to protect redundant resources from common failure modes.

- If a system has two power supplies, either of which can operate the system, what happens when one fails?

  - How is the failure of the first-supply detected?
  - What are the assumed failure modes of the power supply?
  - Can any of these failure modes compromise the integrity or availability of the second power supply?
    One way to evaluate the independence of power supplies is to visualize that each power source outputs a unique color of electrons and each conductor or electronic device that an electron passes through is given that color. Any point where colors mix is a fault propagation path from one power supply to another.

    This technique can be expanded to a form of zonal fault containment analysis where electrons are assumed to be able to jump gaps between conductors if the voltage on the conductor is sufficient to jump the gap. Such analysis should take into account the fact that the gap size that can be jumped is affected by certain failures, geometries, and environmental conditions (e.g., the length that tin whiskers can grow, the distance that an arc-over can jump given variations in air pressure due to altitude changes, any crossovers of adjacent layers in a printed circuit board, adjacent pins in a connector, . . .).

Similarly, in highly-integrated distributed systems, switches may be shared across different system functions, which can render the associated systems vulnerable to common-mode failures.

- When multiple-pole switches or relays are used to select or initiate separate functions, how is functional separation and independence assured?

- Has proper attention been paid to failure modes of the switch or relay that could affect multiple functions?

- Has a benign default system state been defined to address invalid switch or relay inputs?

  Another consideration is that a single fault may result in identical but wrong data. For example, if left and right localizer signals are on adjacent pins, a single short could cause both signals to be identical but wrong. In addition, all users would see the same signals. A failure that affects only one signal could cause loss of system redundancy that cannot be detected, since you cannot tell the good from the bad.

One often overlooked source of error propagation in distributed systems is the communication of floating-point variables. Special care is required to ensure that an erroneous floating-point value does not excite the floating-point error handling routines of receivers. Any assumptions with respect to the assumed validity of the floating-point values will ideally be traceable to fault-containment strategies. Note that unless the transmitter contains self-checking monitored hardware that precludes erroneous value escapes, the erroneous floating-point error handling strategy should be implemented within the receiver logic.

- Do subcomponents of the system communicate using floating-point variables?

- What are the failure assumptions with respect to the floating-point errors?
- What fault containment mechanisms are implemented within clients receiving floating-point values?
  - Does the receiving code implement a Not a Number (NAN) monitor?
  - Are suitable range-limits in place before downstream processing is implemented?
  - What analysis and justification ensures that the receiving system will not incur numerical value underflow and overflow exceptions?
  - Has the time budget for exception handling been accommodated within the receiving system scheduling assumptions?

Issues relating to buffer underflows and overflows should also be given special care. Similar to the above issues with floating-point values, they may invoke error handling routines at the receivers and become a source of error propagation. Note that buffer overflows are a major source of security holes.

- How are buffer overflow and underflow related errors prevented within the system architecture?
- Does the system data-flow incorporate variable size packets or data sequences?
  - If so, how are the lengths and sizes of these packets communicated?
  - What prevents an erroneous length field from escaping?
  - What is the impact of the erroneous length or packet size data?

When error detection encoding (e.g., CRC, parity, checksum) is used to contain errors:

- How was the underlying error behavior determined?
  - What are the probabilities for all the expected error patterns?
  - If the assumption is that bits fail independently, what substantiates this assumption?
    To make this assumption on communication links, there must be negligible inter-symbol interference, no multiple bit encodings (e.g., 4B/5B, 8B/10B), no manipulation of the bitstream between the source's calculation of the error detection code and the receivers' calculation (e.g., no bit stuffing, no compression, no encryption, ...), and no complex integrated circuits (ICs) in the path.
  - How was Bit Error Ratio (BER) [5] testing done?
    Note: BER can only be determined from testing. One cannot cite standards documents; the reference to BER in these documents are requirements, not statements of fact. Citing BER from data sheets can only be done if those numbers were determined from testing in exactly the same environment as the one in which the part is to be used and was done using data with the same or worse characteristics than is expected in this use.
  - What was done to determine possible sources of correlated failures (e.g., crosstalk in communication lines, proximity in memory) and the probabilities of correlated errors? And, what mechanisms are in place to ensure that this question is revisited for any changes in the system during its lifetime (e.g., rerouting of signal wires, die layout changes for ICs in the design)?

[5] It should be noted that BER is also used for Bit Error Rate. The Bit Error Rate and Bit Error Ratio can be related using the bit rate

- How was the coverage of the error detection coding calculated?
- Is this coverage data dependent? If so, what is the worst-case data pattern(s)?

3.4 Design Assumptions, Assumption Rationale, and Robustness

As discussed above, some system failures result from the mismatch of designer's assumptions with respect to the system environment, operation, and fault models. The questions in this area explore this aspect of the system design in more detail.

- What assumptions have you made about the design? Consider:
  - How the design will be operated
  - The physical, operational environment
  - Other systems with which it will interact
  - The performance of components and subsystems (including failure modes and rates of failure)
  - How the system will be maintained and inspected
  - How the system will change over time
- What systems are in place to support these assumptions (i.e., to encourage them to come true)?
- What systems are in place to create, collect, monitor, and react to evidence related to these assumptions?
- Does the system change its behavior (i.e., reconfigure) in response to any observed changes in assumptions?
  - If so, how?
  - Can the reconfiguration be invoked by transient errors?
  - How are transient occurrences differentiated from permanent faults?

  As background to these questions about monitoring possible changes in assumptions, consider the inline CRC and checksum integrity code schemes discussed in the previous section. The application of inline integrity coding assumes a bounded bit error ratio from the communication channel [13]. If the assumed error ratio is exceeded, the integrity assumptions related to the protected data flows may be compromised and reconfiguration of the system to remove the erroneous stream may be prudent. However, this consideration must be balanced against the impact of external, transient disturbances.
- What is the evidence that these systems are suitably trustworthy given the reliance you place on each assumption?
- Under what circumstances would the current claims of safety be invalidated, e.g., unintended uses, extreme environments, etc.?
  - What mechanisms are in place to look for those circumstances, and (provisionally) what action will be taken when they arise?
  - Are there mechanisms in place for robustness (maintaining correct system operation even if the system fault-hypothesis is exceeded)?

  - How did you calculate the probability that the system would exceed its fault-hypothesis?

- For software exception handling:
  - Has a handler been defined for each possible exception?
  - What analysis has been done to ensure that exception handlers don't cause additional exceptions (e.g., time-outs).
  - What analysis has been done to ensure that exception handling doesn't cause deadlocks or livelocks?

3.5 Replication, Redundancy, Input Congruency, and Agreement

Distributed systems often incorporate redundancy to meet system availability and integrity goals. Management of this redundancy largely characterizes the distrusted system logic. Understanding the rationale behind system redundancy is paramount in distributed system design, specifically understanding whether the redundancy is targeted towards integrity, availability, or both. For example, in some systems, multiple sensors may also be cross-compared to implement a sensor diagnosis functions. In such cases, how the failure modes of the sensor or the underlying system communication and computational platform can impact the sensor diagnosis validity must be understood. Similarly, where redundancy is used for integrity, such as the bit-for-bit comparison of the self-checking-paired based schemes, it is important to understand what mechanisms can establish and maintain the state congruency agreement assumed between each half of the pair. The questions below explore these aspects of the design.

- Does the system incorporate redundant sensors?
  - If so, are redundant sensors compared as part of the sensor diagnosis function?
  - Is it possible for an erroneous sensor reading to indict a properly functioning sensor? *As illustrated in Osder [29], without suitable care in the design, it is possible for a faulty sensor to erroneously indict good sensors, needlessly lowering the system redundancy.*

- What analysis has validated the sensor diagnosis logic?
  - What fault model was used during this analysis? Does this fault model fully characterize the failures of the full sensing and communication paths?
  - Does the sensor diagnosis logic take account of the system timing and composition? See sections 3.6 and 3.7 for more details.

- Are redundant hardware components separated such that a failure or likely combination of failures will not cause the loss of all redundant hardware?

- Are redundant input/output (I/O) channels adequately separated? *Answering this question may involve examining input pin spacing and internal wire routing to ensure separation. Redundant I/O that must share a common assembly should be routed to different electronic devices to lessen the probability of a single failure causing loss of all redundancy.*
  - Is a single sensor providing input to redundant I/O?
  - Is wiring routing consistent with the redundancy philosophy?
  - How are bridging faults mitigated?

  - Is redundant signal separation maintained internally as well as externally?
  - What methods and tools were used to determine adequate separation and the risk of losing separation (e.g., "Wire Interconnect System Risk Assessment Tool" [30]).

- Is the rationale for physical separation documented?
  Ideally, the rationale for physical separation requirements should be documented either explicitly or as a derived requirement.

- If any sensor reading is processed by multiple clients:
  - Are there any requirements for having sensor values seen by multiple clients be congruent (each copy of the values are identical or similar enough to meet system requirements)
  - If multiple clients are assumed to receive the same sensor value, what mechanisms are in place to ensure that a congruent sensor value is utilized by all clients?
  - What happens if the sensor clients do not receive the same data?
  - What degree of non-congruent sensor values can be tolerated by the redundancy agreement model?
  - How does a single non-congruent value impact the system differently from the multiple non-congruent values?

The issue of sensor input congruency is part of the larger system theme of computation replication and management. In safety-relevant systems, computation functions are often replicated to provide improved computation availability and/or integrity. Of particular importance for such configurations are the mechanisms and protocols that ensure that the replicated computation functions establish and maintain the required level of agreement with respect to their inputs and internal state variables.

- What level of state agreement is required or assumed among replicated computation functions?
  - How and where is the agreement specified?
  - Why is this agreement needed?
    * Is the replication for availability or integrity or both?
  - What is the impact of this agreement failing?

- What is the design for maintaining this level of agreement?

- What level of data exchange is required to implement the required level of agreement?
  - What is the assumed fault model for the data exchange?
    * Is the assumed fault model for state data exchange consistent with the implementation of the system and fault-hypothesis and fault-containment mechanisms?
  - How does the system temporal composition impact the state agreement exchange? See section 3.6.
  - Is system bandwidth and/or system scheduling analysis provided for the data exchange required for agreement?

- What analysis demonstrates the validity of the state-exchange and agreement logic under the assumed fault-hypothesis?
- For redundancy patterns that use leader election, how is the initial leader selected?
- For active-standby configurations, how is the active node initially arbitrated or selected?
- What analysis validates the correctness of the leader-election or active-standby arbitration scheme in conjunction with the system temporal composition, communication latencies, and fault-hypothesis? See sections 3.2 and 3.6.
- What is the system-level impact of the leader-election or active-standby logic failing?
  - Can the system detect whether there is more than one group leader (e.g., when a backup processor spuriously detects that the active leader is down when in fact the active leader is still working)?
  - How is this scenario resolved? Under what circumstances can the mechanism that resolves this scenario cause a scenario where no leader is elected?
- Given a choice between solving the scenario of "no leader" versus solving the scenario of "multiple leaders", which is more important and why?

Architectures for fault-tolerant systems can be designed such that their processing elements and the communication interfaces between these processing elements have different guaranteed failure modes. Some system designs, such is the Boeing 777 Aircraft Information Management System (AIMS) [31] enforce a restriction that allows only benign failure modes (e.g., fail-crash, fail-silent, fail-stop) for both the processing elements and their communication interfaces. Other system designs, such as the Delta-4 [32] architecture, allow a mix of the benign and non-benign failure modes of its processing elements, but require benign-only failure modes of its communication interfaces. And, still other system designs, such as Scalable Processor-Independent Design for Electromagnetic Resilience (SPIDER) [33, 34], Multicomputer Architecture for Fault Tolerance (MAFT) [35], and BRAIN [28] allow non-benign failures of both its processing elements and their communication interfaces. In addition to these differences, architectures can have different requirements on the relationship between the outputs of redundant components. The output values may be required to be bit-for-bit-identical (typically followed by M-out-of-N voting or exact-match comparison for error detection) or may be allowed to be different by some bounded degree (typically followed by a mid-value selection, "force fight", or summing). The times when these outputs are produced may be constrained to have a tight skew between redundant components or may have looser time constraints. Various methods may be employed for enforcing any required relationship between redundant outputs, in both value and timing. These methods can include congruency exchanges, peer equalization, master-slave state following, or other techniques.

- What failure modes of processing elements can this system architecture tolerate?
- What are the mechanisms that implement this tolerance?
- What analysis exists to justify that failure modes of processing elements that the higher-level architecture cannot tolerate have a sufficiently low probability of occurrence?
- What failure modes of communication interfaces can this system's architecture tolerate?
- What are the mechanisms that implement this tolerance?

- What analysis exists to justify that failure modes of the communication interfaces that the higher-level architecture cannot tolerate have a sufficiently low probability of occurrence?
- What mechanisms enforce any relationship requirements between the values of redundant outputs?
- What analysis has been done to show that the range of values constrained by these mechanisms meets system requirements?
- What mechanisms enforce any relationship requirements between the timing of redundant outputs?
- What analysis has been done to show that the timing constrained by these mechanisms meets system requirements?
- How is the required level of agreement among redundant computation functions established at system startup? See section 3.10 for related questions.
- How is agreement established following node re-integration?
  - How is the level of agreement among redundant computational functions affected by system modes and power transients?
  - What analysis justifies the above claims?
  - What fault model was used as part of this analysis? See section 3.2.

3.6 Temporal Composition and Determinism

In distributed systems, the system timing and associated communication models play a significant role with respect to characterizing the system behavior and fault-tolerance. Modeling and analyzing the communication primitives of the system can be greatly beneficial. The questions below explore this aspect of system design.

- How is the system temporal composition defined?
- Is the underlying model of the system redundancy management and data flow synchronous, asynchronous, or a mix of both?
- How are task precedence constraints specified and analyzed?
- What is the impact to the system if these task precedence constraints are not honored?
- If the system is (partly) asynchronous, how are the required agreement properties discussed in section 3.5 achieved?
  - Which asynchronous consistency/consensus algorithms/protocols are you using?
  - How do you guarantee the correctness of these approaches (e.g., formal methods)?
  - Is the fault model assumed by such analysis consistent with the system fault-hypothesis?
- If the system is (partly) synchronous, how do you ensure the safety and availability of synchrony (e.g., fault-tolerant clock synchronization)?
  - How do you guarantee the correctness of these approaches (e.g., formal methods)?

  - Is the fault model assumed by such analysis consistent with the system fault-hypothesis?
  - How does startup affect the analysis?

- If the system is (partly) asynchronous, how do you calculate the real-time characteristics of the system components?
  - For networks, is network calculus used?
  - Is the fault model used for such analysis consistent with the system fault-hypothesis?
  - For nodes, is response-time analysis used?

- For both synchronous and asynchronous systems, how is the impact of non-deterministic mechanisms bounded? Typical sources may include:
  - Waiting for hardware response, such as:
    * Waiting for EEPROM to not be busy before writing an error code
    * Waiting for serial communication device to finish sending (or clear buffer) before writing next byte
    * Polling user interface keys, waiting for stable transitions (debouncing, etc.)
    * Coming out of low-power or standby modes
  - Dynamic memory allocation (and garbage collecting, if applicable)
  - 3rd-party library routines with no timing information
  - Processor cache misses, pipeline flushes, incorrect branch prediction guesses, out-of-order instruction execution
  - Unbounded recursion

- If the system comprises both globally synchronously scheduled traffic together with asynchronous locally scheduled traffic, what have you done to bound the interference between the two traffic classes?
  - Does your analysis accommodate all anticipate traffic scenarios such as system start-up, and clique recovery?

3.7 Latency

The impact of communication latency within distributed systems also needs careful consideration. The next set of questions explore the issues relating to latency in more detail.

- Has data latency been addressed by a worst-case timing analysis?

- Have response delays from interfacing subsystems been considered in the latency analysis?

- Have transport delays (from pilot selection to system response) been considered in the system latency analysis?

- Can data be missed without any effect on system operation?
  - If so, which data? How much data, when, etc.?
  - What analysis justifies these assumptions?

- Can the system operate in a degraded mode with loss of some data? If so, which data, how much data, when, etc.?

- Have degraded modes of system operation been defined for cases where less than all data is available?
- With respect to multi-path and multi-party communication, what assumptions exist for the distribution of data?
  - For example, are all clients on a bus assumed to get congruent data?
  - What is the impact to the system if a non-congruent data broadcast occurs; that is, do some parties on the bus get the correct data while others get incorrect or no data?
    Note that this aspect is particularly important for receivers that have been replicated for integrity comparison. See section 3.5.

3.8 System Interfaces and Environment

It is very important to understand the system boundary and external system dependencies, particularly for distributed systems that may include many separate sub-system interface points and dependencies. It may be beneficial to examine and document the rationale and assumptions behind inter-system dependencies, which often occur at the edges of the system where multiple organizations are interfacing and where non-documented or non-congruent assumptions exist.

- How have you ensured that the system boundaries and the environmental conditions are known, at least to a sufficient degree?
- Are interfacing systems aware of the system failure modes?
  - How are these failure modes communicated between design teams?
- Does the system assume dependencies of the external system components?
  - For example, does the system assume that power is applied in a particular order?
  - What has been done to ensure that this assumption is valid?
    * If this order is not observed, does system fault tolerance degrade?
      An interesting example of such a dependency is the TTP protocol, which requires the central guardian components to be operational before the startup of the hosting nodes. If this dependency is not observed, it may be possible for a faulty guardian to establish cliques during startup.
- What is the effect of system failures on interfacing systems?
- What has been done to ensure that the OEM and supplier have congruent, aligned assumptions relative to the system architecture and its associated fault model?
- In addition to testing for individual environmental conditions, such as defined in DO-160 [36], what combinations of environmental conditions have been tested?
  For example, pressure differential and temperature [27, p. 38] or complex scenarios such as "a climbing right-hand turn of more than 3 Gs above 10,000 feet" [27, p. 24]
- Is the system architecture hosting third party applications?
- How have you communicated the fault-model and platform availability characteristics to the third parties?

  - How are the fault assumptions of third-party-hosted applications ratified against the fault models of the hosting platform?

- What measures have you taken to establish the safety integrity of third-party and legacy software?

3.9 Shared Resource Management

In many safety-relevant systems, the management of shared resources is a critical aspect of the system design. Special care is needed to ensure that the system is not vulnerable to deadlock, resource starvation, or live-lock. The Mars lander "Mars Pathfinder" is a classic example of a deadlock caused by priority inversion in a realtime systems [37]. In distributed systems, this is further complicated by the distributed nature of the mutual exclusion and resource management algorithms. The following questions are intended to explore issues relating to local and global resource management.

- If the system uses shared resources, what system mechanisms are deployed to manage the shared resources?
- What are the fault assumptions of the resource management logic?
  - Are all parties assumed to follow the resource management protocol correctly?
  - Does the implementation platform that is hosting the resource management support the assumed fault model?
- What happens if a function fails without releasing a resource or fails during a critical section?
- Are resource management actions assumed to be atomic?
- What mechanisms are in place to ensure these atomic actions?
- What analysis and/or mechanisms are in place to prevent starvation?

3.10 Initialization and System Startup

In distributed systems, system initiation and startup are particularly problematic because the system may lack the number of required resources to form voting planes or similar fault-containment strategies at startup. At the beginning of startup, it may appear that all the system components have failed. Given such a fault model, forming the initial agreement (either bounded or exact) among redundant components can be very challenging and is often a weak point of distributed systems.

- Has default data been defined for inputs that are invalid or missing at startup?
- Have requirements been defined for any default data used at startup?
- Have default data been defined for inputs that become missing (e.g., a rejected message due to bad CRC) or invalid after initialization?
- Have requirements been defined for use of default data for missing or invalid inputs after initialization?
- If the initial default data is defined by hardware (e.g., all registers cleared on reset):

  - Are these default values set correctly for both power-on reset and the application of reset signals [27, p. 41]?
  - If the hardware mechanism for setting the default values is not explicitly documented in the device's data sheet, what mitigations exist to ensure that the system works correctly if this mechanism for setting default data no longer works (e.g., future undocumented IC die "improvements")?

- What requirements have been defined for power transients?
- Has synchronization of multiprocessors been considered? See sections 3.5 and 3.6 for additional questions.
- Are data reasonableness checks performed wherever appropriate (i.e., calculation of flight phase)?
- Do the power transient requirements reflect all possible scenarios to which the system may be subjected?
- Has definition of state variables and checkpoint methodology been completed?
  - What analysis justifies the selection of the critical state variables?
- What analysis exists to validate system check-pointing and recovery protocols?
  - Does this analysis cover all of the anticipated operational scenarios?

3.11 Modeling and Analysis

System modeling can be an effective method for exploring and understanding system properties. Suitable models may allow the exploration of extreme conditions that are not possible or too expensive using real system hardware. However, for all models, the intent of the modeling effort must be clear, and the abstraction of the models and how they differ from the real system environment must be well understood.

- What level of modeling has been performed to validate system requirements?
- What design questions need to be answered by your models and why are these relevant to safety?
- Which safety relevant question cannot be answered with modeling?
  - Why not?
  - What alternative method(s) did you use?
- How did you ensure that each model's semantics were correctly understood?
- Does the system modeling make it possible to explore extremes of the system temporal composition (see section 3.6) and fault-hypothesis (see section 3.2)?
- Does the system model enable analysis of higher-level system interactions such as power-on sequencing, power-down modes?

  - Have the higher-level system interaction scenarios been reviewed by system stake-holders and domain experts?

- How have you modeled the application domain? To what fidelity?
- Were domain experts used for this modeling?
- How have you ensured that your models are adequate?
- How have you ensured that your models are kept consistent throughout the whole development cycle?

3.12 Application of Formal Methods

The application of formal-methods-based techniques can be invaluable to distributed system design. These techniques can be effective tools for discovery of system failure *edge cases* that may not be obvious to the designer. They are especially valuable for distributed systems where the complexities of concurrent interactions are difficult for the average human mind to reason about. Formal models also can be used to drive architectural distributed system test generation as detailed in [38].

However, the application of formal-methods techniques is not a silver bullet. Great care needs to be taken to ensure that the abstractions and assumptions of formal models used for analysis match the implementation and environment of the real system. Similarly, where formal proofs for system components (such as protocols) exist, it is important to understand the constraints and context for such proofs to ensure that they match the environment.

- How are the abstractions of the formal model validated?
  - Have they been reviewed and accepted by domain experts?
  - How does the fault model abstraction of the formal proof differ from real-world system fault assumptions? See section 3.2.
  - How does the timing abstraction of the formal proof differ from the real-world system composition? See section 3.6.
  - What makes the temporal assumptions of the formal model sufficient for the inference that is being drawn from the models?
    For example, if the intent of the formal model is to prove system synchronization properties and the model is synchronously composed, some of the issues related to asynchronous interactions may be missed.
  - What behavior is assumed by the formal model?
    * How does system initialization affect the protocol proofs? See section 3.10
    * How does system power-on ordering affect the protocol proofs?
    * Do any software dependencies affect the applicability of the proof?
      Consider the impact of any life-sign strobing (device on a network connection needs to perform an action to the network interface to show it is alive). How are these behaviors impacted within the target system?
- Have the models used for formal analysis passed the well-formed checks supported by the tooling?
  - Are they assured to be dead-lock free?
  - Do they return counter examples for expected scenarios?

3.13 Numerical Analysis

A large part of any dependable digital system involves numerical computations. The following questions deal with numbers and numerical algorithms.

- What analysis has been done to ensure that critical variable values have the needed precision?
- What analysis has been done to ensure that critical variable values stay within acceptable limits?
- For values that are transferred among subsystems, are value limits and precision acceptable for all subsystems? Are limits checked by all subsystems?
- What analysis been done for variable units and dimensionality?
- For all values that need units conversion, have these conversions been double checked?
- How is underflow handled? Is this handling appropriate for all variables?
- How is overflow handled? Is this handling appropriate for all variables?
- How is rounding handled? Is this handling appropriate for all variables?
- For algorithms that need to converge:
  - Does the algorithm converge for all possible inputs?
  - Does the algorithm converge fast enough under all conditions?
  - For algorithms that need a "close enough" starting value to converge, what analysis has been done to show that the starting value always is close enough, particularly during startup?
- What numerical stability analysis has been done?
- For variables that are arrays, vectors, or scalars that take up more than one memory location:
  - Are all the operations to load and store these values atomic?
  - If not, what are the possible consequences of non-atomic updates?
    For an example of what can go wrong, see "Floating Point in Interrupt Handling Routines" [27, p. 46]
- Are all outputs from integrators, filters, and counters amplitude limited?
- For variables that can be written from different sources (particularly different tasks), what analysis has been done to show that the updates from different sources do not cause problems? In particular, have all possible timing relationships of these updates been considered for both normal and abnormal scenarios? How was this analysis performed?
- What analysis has been done to show that the input to trigonometric functions are always valid?
  - Acos input is in the range [-1 .. 1]?
  - Asin input is in the range [-1 .. 1]?
  - Atan2 inputs are not both zero

- For tangent functions, is the appropriate range of input angle used relative to input angles near multiples of $pi/2$?

- Is an *nth* even root taken only of a positive numbers?

- Is the exact equality operator ever used on floating-point numbers?

  Because floating-point representation and operations allow the accumulation of roundoff error, it is necessary to define equality test operators and functions that are cognizant of the upstream error budgets.

3.14 System Test

System testing is a vital part of system verification and validation. Similar to system modeling, it is important to understand the rationale for each test campaign. Another consideration is how the visibility of the target environment may affect the test observations. This is especially true for fault-tolerant distributed systems that often mask fault behavior. In such systems, additional data logging may need to be added within the system design to support the levels of visibility required to support validation objectives. Hence, it is important to establish a clear system test philosophy early in the design cycle.

It is also important to establish clear completeness criteria for system testing. The questions below explore the issues in this area.

- What percentage of the system fault tolerance mechanisms has been exercised within the system test campaign?

- What cannot be tested?

  - Why?
    Note sometimes it is difficult to instrument and configure the required level of fault-injection or monitoring. In such cases, targeted analysis may be useful.
  - What analysis has been performed in place of system test?

- What level of observation ensures that you can monitor the fault-tolerance mechanisms of the architecture?
  Note as discussed above, it is common for fault-tolerant systems to mask error events. Without suitable provision within the test and monitoring framework it may not be possible to fully observe the behavior of the fault tolerance mechanisms. In distributed systems that incorporate distributed fault-tolerance algorithms, this challenge may be further complicated by the distributed state of the fault-tolerance logic.

- Has any unexplained unexpected behavior been discovered during system analysis or testing?
  Unexpected behaviour discovered during testing can reveal fault-tolerance vulnerabilities or unanticipated failure modes. Ideally, there should never be any unexplained lab-testing scenarios. Lab notebooks should be kept that document all unanticipated behavior. Explanations for all such behavior should be found and documented.

- What level of robustness testing has been performed or planned?

- What are your criteria for selecting a robustness test?

- Does the robustness testing include scenarios such as
  - Disconnect and reconnect network cabling from a random selection of points in your system?
  - Simulated power failure and brownouts?
  - Loss of system cooling?

3.15 Configuration Management

By design, distributed systems comprise a set of distributed resources. It is common to support the individual reprogramming of the resources in the field. Thus, it is important for the distributed system to implement functions and protocols to ensure that a valid system configuration is established and maintained.

- To what degree does your system depend on the accuracy of configuration data.
- What measures does your design take to secure the integrity of the configuration data?
- What mechanisms and protocols are in place to ensure that a valid system configuration exists?
- How is a valid system configuration defined?
- For systems integrating subsystems from multiple suppliers, how is the system configuration managed across the organizational boundaries?
  - Who has the responsibility for the integrated system configuration?
- How many configuration faults can the system tolerate?
  - Note: If the answer to this question is none, see the malicious fault discussion in section 3.2. What is different with respect to the assumed configuration failure versus the malicious fault assumption?
- Can any type of misconfiguration faults be undetected?
  - What is the impact of such faults?
  - Do they introduce a reduction in the system safety margins?
    For example, if the temporal arrival windows for data acceptance are artificially small, there may be no observable impact if the system clocks are tightly synchronized. However, as the clock drift degrades with temperature and age, the artificially small windows may erroneously impact system data flow, rejecting good data.
  - How are such scenarios mitigated?
- If a single component suffers a configuration failure, how can it impact the other correctly configured system components?
  Note when performing this analysis it is important to consider the system startup. If the first node powered-up and active is erroneous, it can hold off other well-configured nodes from integrating.

3.16 Operation and Maintenance

The operational and maintenance aspects of system life-cycle are crucially important to safety-relevant systems. This is particularly important for aerospace systems where dispatchability requirements of the aircraft may require the system to operate with known faults present.

- What design measures have you taken to prevent unsafe system maintenance from destroying the safety integrity of your system?
- How does the redundancy management scheme of the system affect dispatch criteria?
  - How are LRU reliability and aircraft dispatch reliability related?
  - Is this relationship clearly documented?
  - Are there any undocumented assumptions?
- What assurances for dispatchability are required after maintenance?
  - What must the return-to-service tests do to assure high confidence in dispatchability?
- Is it possible to install the equipment incorrectly?
  - What techniques have been used to ensure proper installation?
  - If these techniques fail, what is the effect on the aircraft and on the equipment?
- If improper installation can damage the equipment, is the damage and resultant degraded functionality obvious and easily detectable as part of the normal system safety procedures?
- How much of the design depends on human intervention to mitigate safety hazards in high stress emergency environments?
- If you were to evaluate the attitudes and experience of the people responsible for ongoing operations and maintenance:
  - What is their attitude to safety?
  - How many hours of safety training have they had?
  - How many years of safety-related system experience have they had?

Note: The questions for system configuration of section 3.15 also need to be considered within the context of maintenance actions.

3.17 Other Known Design Pitfalls and Caveats

- Which components have errata sheets?
  - Which of them have you read?
  - Have system assumptions been updated in response to the content of errata sheets?
- How are various types of errors handled?
- Is the error handling philosophy consistent throughout the design?
- How do you ensure that error handling produces no unintended effects?
  (For example, using the wrong memory locations.)
- Does the design use self-modifying code or any adaptive features?

- How have you shown that priority inversion is not a problem?
- How do you prove that task scheduling meets all requirements?
- How were the effects of errors due to numeric scaling shown to be acceptable?
- If software is re-used or imported, how have you ensured that it is adequate for the current design?
- Are there interrupts other than Clock Tick and Fatal (an interrupt from which there is no return except through reset)? If so, show the analysis of all their possible interactions.
- When implementing state machines, are there recovery mechanisms for all illegal state values?
- How are the following hardware issues handled?
  - Initialization
  - Noise on sensor (and other) inputs
  - Power up and power down behavior
  - Power usage (sleep) modes
  - Watchdog timer
  - ADC and DAC turn-on delays
  - Flash and/or EEPROM interfaces
- How to you demonstrate that RAM, ROM, and EEPROM sizes are adequate?
  - Are the stack and heap adequately sized for worst-case scenarios?
- How have the following been handled, in value, time, and space?
  - Intermediate data
  - Data shared between interrupt service routines and application
  - Data shared between tasks of different priorities
- For partitioned processors, how do you know that there are no mechanisms where a low criticality partition can adversely affect a higher criticality partition?

3.18 Organizational Factors

Understanding the organization factors that influence the distributed system design are also important.

- Who is personally accountable if the design proves not to be safe?
  - What is this person's involvement in design reviews?
  - Has the individual signed to accept accountability?
  - Has a director of the organization signed to accept liability?
- Has the design team been furnished with the relevant Advisory Circulars?
- Has the design team been adequately trained with respect to the recommended practice for safety-relevant functions [9, 22]?
- Can you provide some background on the people involved in the process?

- Can you present evidence that they are capable of recognizing credible hazards?
  The ideal review team should comprise as many years of domain experience as possible.
- What relevant lessons learned or heuristics, either within the companies that develop the product or within the engineering community, have been applied?
- If applicable, what are your plans for working with others who are designing the overall system.
- If applicable, what are your plans for working with a distributed design team?
- What configuration controls do you have in place? And, how are they audited?
  See section 3.15 for additional questions relating to platform configuration control.
- Are material lots or batches (IC dies from the same wafer or batch, printed wiring boards from the same table, heat sink glue from the same batch, etc,) tracked so that all faulty parts can be identified if there is a process failure that causes generic failures in the product?

3.19 Certification

System certification is a complex multifaceted topic requiring extensive coordination with a DER and a detailed certification plan. The following questions constitute some high level considerations that may be generally applicable. This set of questions is just an initial start. It is hoped that a more thorough set of question can be developed through industry engagement on NASA DASH*link*[1].

- How have you ensured compliance with any applicable prevailing technical standards or regulations—including safety standards?
- Does the system have an agreement and released certification plan? If not, when is the completion date?
- Is the system subject to maturity requirements that force an early completion of the design, implementation, and V&V processes?
- Is there an existing advisory circular or TSO for the product being designed, or is there risk that one will appear late in the program?
- Is there a requirement to certify ground-based or off-line tools?

4 Discussion and Future work

4.1 Public Dissemination

This work has correlated an initial set of checklist questions for safety-relevant, distributed system design. As emphasized in the introduction, we do not claim that the initial set of questions presented herein is complete. As new lessons are learned and new perspectives are added, new questions will need to be added to this initial question set. It will be particularly valuable if the added questions can be related to additional stories of how systems can fail.

To support continuation of this exercise, we suggest posting the checklist to a public server. The feedback we received from public safety mailing lists and groups [18] [19] gives us confidence that, suitably advertised, such a site will attract and benefit from the external support. We suggest that, once the checklist is added, its location is advertised to additional groups such as LinkedIn and SAE safety-related working groups.

To support this goal, a set of questions formatted in HTML has been delivered with this report. The mind-map from TheBrain tool can also be exported to simple and application-script assisted HTML formats. An initial version of the checklist mind-map in both brainzip [21] and HTML formats also accompanies this report.

4.2 Potential Research Expansions

During the production of this work, we correlated the input and questions from many different sources. In many places, we found redundancies as the same question was asked in different ways. From this experience, we postulate that it may be possible to establish a metric for the distributed system design and rationale consistency. For example, it is possible to develop a set of targeted questions asking the same questions from an availability or integrity system bias, to assess the balance of the system design rationale. Might it be possible to generate a Design Type Indicator, similar to the Myers-Briggs Personally Type Indicator [39] for the distributed systems? Additional aspects of such a metric may include design rationale consistency, prevalence of an integrity versus an availability mindset, optimism versus pessimism about failure mode assumptions, and so on.

Another idea is to integrate a variant of this question set with an expert system to support a semi-automated, guided design review. If such a strategy is integrated with formal descriptions of the architectures, such as suitably annotated AADL models, we believe it will be possible to develop a set of interactive questions targeted to particular architectural patterns and frameworks.

5 Terms, Abbreviations, and Glossary

AADL	Architecture Analysis and Design Language
ADC	Analog to Digital Converter
AIMS	Aircraft Information Management System
AFCS	Assurance of Flight Critical Systems
API	Application Programming Interface
ARINC	was Aeronautical Radio INC., no longer an acronym
BER	Bit Error Ratio
BRAIN	Braided Ring Availability Integrity Network
CM	Computation Module
CRC	Cyclic Redundancy Check
DAC	Digital to Analog Converter
DER	Designiated Engineering Representative
EEPROM	Electronically Erasable Programmable Read-Only Memory
FAA	Federal Aviation Administration
FACE	Future Airborne Capability Environment
FADEC	Full Authority Digital Engine Control
FCR	Fault Containment Region
FMEA	Failure Modes and Effects Analysis
FMS	Flight Management System
HTML	Hyper Text Markup Language
IC	Integrated Circuit
IFIP	International Foundation for Information Processing
IMA	Integrated Modular Avionics
LRU	Line Replaceable Unite
NAN	Not a Number
NASA	National Aeronautics and Space Administration
MAFT	Multicomputer Architecture for Fault Tolerance
OASM	Output Actuation Sense Module
OEM	Original Equipment Manufacture
RAM	Random Access Memory
ROM	Read-Only Memory
SAE	Society of Automotive Engineers
SAL	Symbolic Analysis Laboratory
SPIDER	Scalable Processor-Independent Design for Electromagnetic Resilience
TSO	Technical Standard Order
TTP	Time Triggered Protocol
VL	Virtual Link, a preconfigured, enforced network route

6 References

References

1. Assurance of Flight Critical Systems (AFCS).
   https://c3.nasa.gov/dashlink/projects/79.

2. Dvorak, D.; et al.: NASA study on flight software complexity. *NASA Office of Chief Engineer*, 2009.

3. Judas, P. A.; and Prokop, L. E.: A historical compilation of software metrics with applicability to NASAs Orion spacecraft flight software sizing. *Innovations in Systems and Software Engineering*, vol. 7, no. 3, 2011, pp. 161–170.

4. Itier, J.-B.: A380 integrated modular avionics. *Proceedings of the ARTIST2 meeting on integrated modular avionics*, vol. 1, 2007, pp. 72–75.

5. Ananda, C.: General aviation aircraft avionics: Integration & system tests. *Aerospace and Electronic Systems Magazine, IEEE*, vol. 24, no. 5, 2009, pp. 19–25.

6. Prisaznuk, P. J.: Arinc 653 role in integrated modular avionics (IMA). *27th IEEE/AIAA Digital Avionics Systems Conference (DASC)*, IEEE, 2008, pp. 1–E.

7. Rushby, J.: Partitioning for Avionics Architectures: Requirements, Mechanisms, and Assurance. NASA Contractor Report CR-1999-209347, NASA Langley Research Center, June 1999.
   http://www.csl.sri.com/users/rushby/abstracts/partitioning.

8. Future Airbourne Capability Environment.
   http://www.opengroup.org/FACE/vision-and-mission.

9. SAE: Aerospace Recommended Practice ARP4754, Revision A: Guidelines for Development of Civil Aircraft and Systems.
   http://standards.sae.org/arp4754a/, 2010.

10. RTCA: RTCA DO-297: Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations.
    http://www.rtca.org/store_product.asp?prodid=617, 2005.

11. RTCA: RTCA DO-178C: Software Considerations in Airborne Systems and Equipment Certification.
    PDF: http://www.rtca.org/store_product.asp?prodid=803
    Paper: http://www.rtca.org/store_product.asp?prodid=914, 2011.

12. RTCA: RTCA DO-254: Design Assurance Guidance for Airborne Electronic Hardware.
    PDF: http://www.rtca.org/store_product.asp?prodid=752
    Paper: http://www.rtca.org/store_product.asp?prodid=753, 2000.

13. Koopman, P.; Driscoll, K.; and Hall, B.: Selection of Cyclic Redundancy Code and Checksum Algorithms to Ensure Critical Data Integrity. FAA Contractor Report DOT/FAA/AR-xx/xx [number not yet assigned], U.S. Department of Transportation, Federal Aviation Administration, Sept. 2013.

14. ARINC: Specification 825-2, General Standardization of CAN (Controller Area Network) Bus Protocol for Airborne Use.
https://www.arinc.com/cf/store/catalog_detail.cfm?item_id=1642, 2011.

15. ARINC: ARINC 664P7-1 Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet (AFDX) Network. Jun 2005.
https://www.arinc.com/cf/store/catalog_detail.cfm?item_id=1270.

16. Japan Automotive Software Platform and Architecture - Functional Safety working group.
https://www.jaspar.jp/english/guide/wg_fs.php.

17. Driscoll, K.: Real System Failures.
https://c3.nasa.gov/dashlink/resources/624, Sep 2013.

18. The System Safety Mailing List.
http://www.systemsafetylist.org.

19. International Federation For Information Processing.
http://www.dependability.org/wg10.4.

20. Driscoll, K.; Hall, B.; Koopman, P.; Ray, J.; and DeWalt, M.: Data Network Evaluation Criteria Handbook. FAA Handbook DOT/FAA/AR-09/24, U.S. Department of Transportation, Federal Aviation Administration, Jun 2009.

21. TheBrain Technologies LP: *The Brain7 Users Guide*. 2012.
http://assets.thebrain.com/documents/TheBrain7-UserGuide.pdf.

22. SAE: Aerospace Recommended Practice ARP476: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment.
http://standards.sae.org/arp4761/, 1996.

23. SAE: SAE Standard AIR6110 Contiguous Aircraft/System Development Process Example.
http://standards.sae.org/wip/air6110, n.d. Work in progress, 2013.

24. SAE: S-18, Aircraft and Sys Dev and Safety Assessment Committee.
http://www.sae.org/works/committeeHome.do?comtID=TEAS18.

25. Lamport; Shostak; and Pease: The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, vol. 4, July 1982, pp. 382–401.

26. Driscoll, K.; Hall, B.; Sivencrona, H.; and Zumsteg, P.: Byzantine Fault Tolerance, from Theory to Reality. *Computer Safety, Reliability, and Security*, G. Goos, J. Hartmanis, and J. van Leeuwen, eds., Lecture Notes in Computer Science, The 22nd International Conference on Computer Safety, Reliability and Security SAFECOMP, Springer-Verlag Heidelberg, September 2003, pp. 235–248.

27. Driscoll, K.: Real System Failures: Observed Failures Scenario.
https://c3.nasa.gov/dashlink/static/media/other/ObservedFailures1.html, Sep 2013.

28. Driscoll, K.; Hall, B.; and Varadarajan, S.: Maximizing fault tolerance in a low SWaP data network. *31st IEEE/AIAA Digital Avionics Systems Conference (DASC)*, 2012, pp. 7A2–1–7A2–16.

29. Osder, S. S.: Generic Faults and Architecture Design Considerations in Flight Critical Systems. *AIAA Journal Of Guidance*, vol. 6, no. 2, March–April 1983, pp. 65–71.

30. Linzey, W. G.: Development of an Electrical Wire Interconnect System Risk Assessment Tool. FAA Contractor Report DOT/FAA/AR-TN06/17, U.S. Department of Transportation, Federal Aviation Administration, Sept. 2006.

31. Driscoll, K. ; Hoyme, K.: The Airplane Information Management System: an integrated real-time flight-deck control system. *Real-Time Systems Symposium*, 1992, pp. 267–70. `http://ieeexplore.ieee.org/iel2/447/6239/00242654.pdf`.

32. Chereque, M.; Powell, D.; Reynier, P.; Richier, J.-L.; and Voiron, J.: Active replication in Delta-4. *Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on*, 1992, pp. 28–37.

33. Torres-Pomales, W.; Malekpour, M. R.; and Miner, P.: ROBUS-2: A Fault-Tolerant Broadcast Communication System. NASA/TM-2005-213540, NASA Langley Research Center, 2005.

34. Latronico, E.; Miner, P.; and Koopman, P.: Quantifying the Reliability of Proven SPIDER Group Membership Service Guarantees. *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, DSN '04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 275–284. `http://dl.acm.org/citation.cfm?id=1009382.1009742`.

35. Keichafer, R.; Walter, C.; Finn, A.; and Thambidurai, P.: The MAFT architecture for distributed fault tolerance. *Computers, IEEE Transactions on*, vol. 37, no. 4, 1988, pp. 398–404.

36. RTCA: RTCA DO-160G: Environmental Conditions and Test Procedures for Airborne Equipment.
PDF: `http://www.rtca.org/store_product.asp?prodid=770`
Paper: `http://www.rtca.org/store_product.asp?prodid=781`, Aug 2011.

37. Reeves, G. E.: What really happened on Mars? `http://research.microsoft.com/en-us/um/people/mbj/mars_pathfinder/authoritative_account.html`, Dec. 1997.

38. Hall, B.; Dietrich, P.; and Driscoll, K.: Model-Driven Test Generation for Distributed Systems. Contractor Report NASA/CR-2013-xxxx, NASA Langley Research Center, Langley Research Center, Hampton VA 23681-2199, USA, Sep 2013. To be released.

39. Myers, I. B.; McCaulley, M. H.; and Most, R.: *Manual: A guide to the development and use of the Myers-Briggs Type Indicator*. Consulting Psychologists Press Palo Alto, CA, 1985.

Appendix A

Appendix : Checklist Mind Map

This figure below illustrates a top-level context of the architectural checklist mind-map. The intent of the mind-map is to support the questions, using the examples of real-world failure scenarios. The content of the mind map can be viewed using TheBrain [21] from Brain Technologies. The *brainzip* and an HTML-based rendering of this mind map will be posted to the NASA DASH*link* AFCS repository [1] .

Figure A1 shows a top view the mind map with the focus at the top level of the questions.

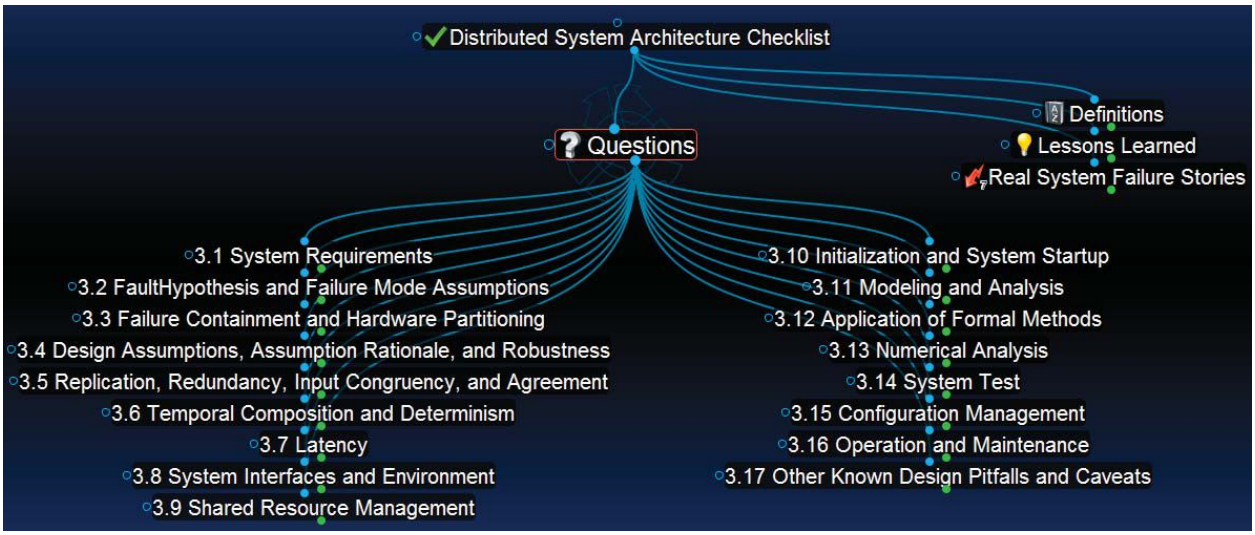

Figure A1. Top-Level Mind Map with Focus on Questions

Figure A2 shows a second view of the mind map with the focus at the top-level of the stories.

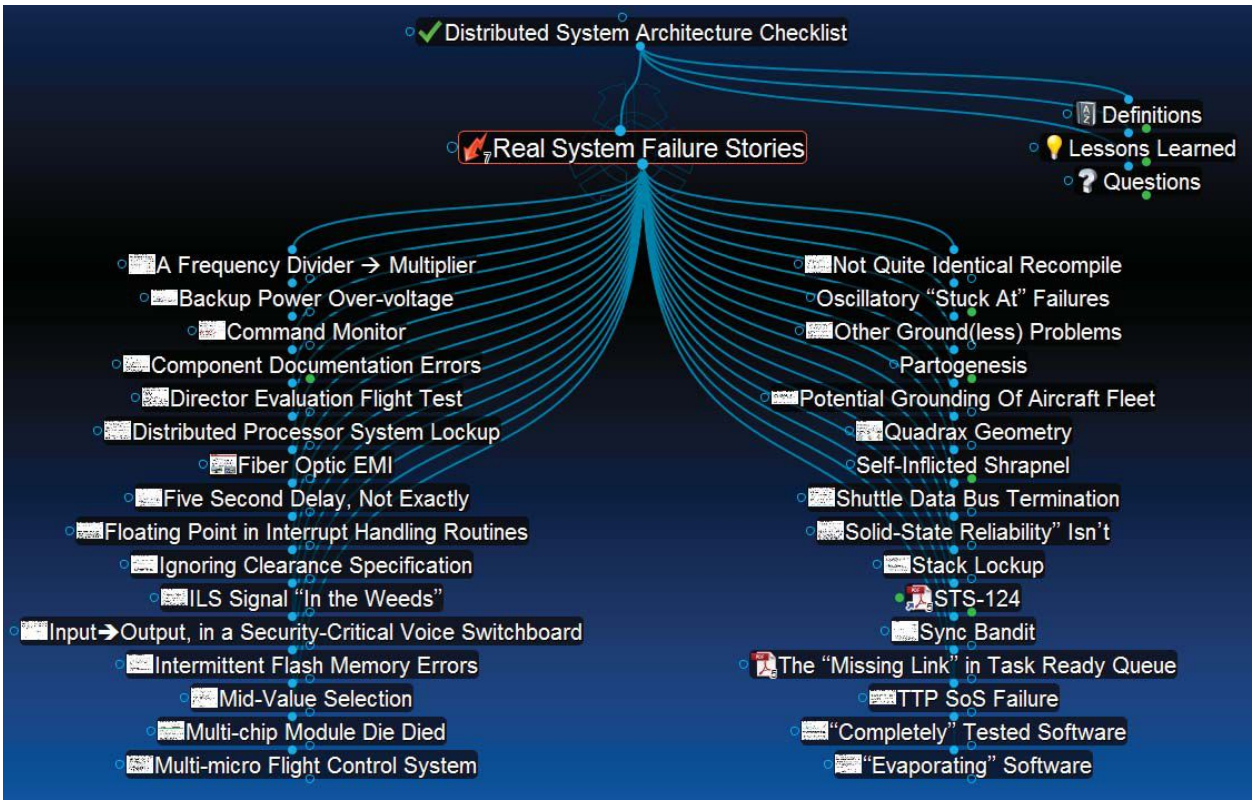

Figure A2. Top-Level Mind Map with Focus on Stories

Figure A3 shows a final view of the mind map that illustrates how the stories can be linked to the questions and vice-versa. The links highlighted using yellow dots, denote the linkages between the questions and stories.

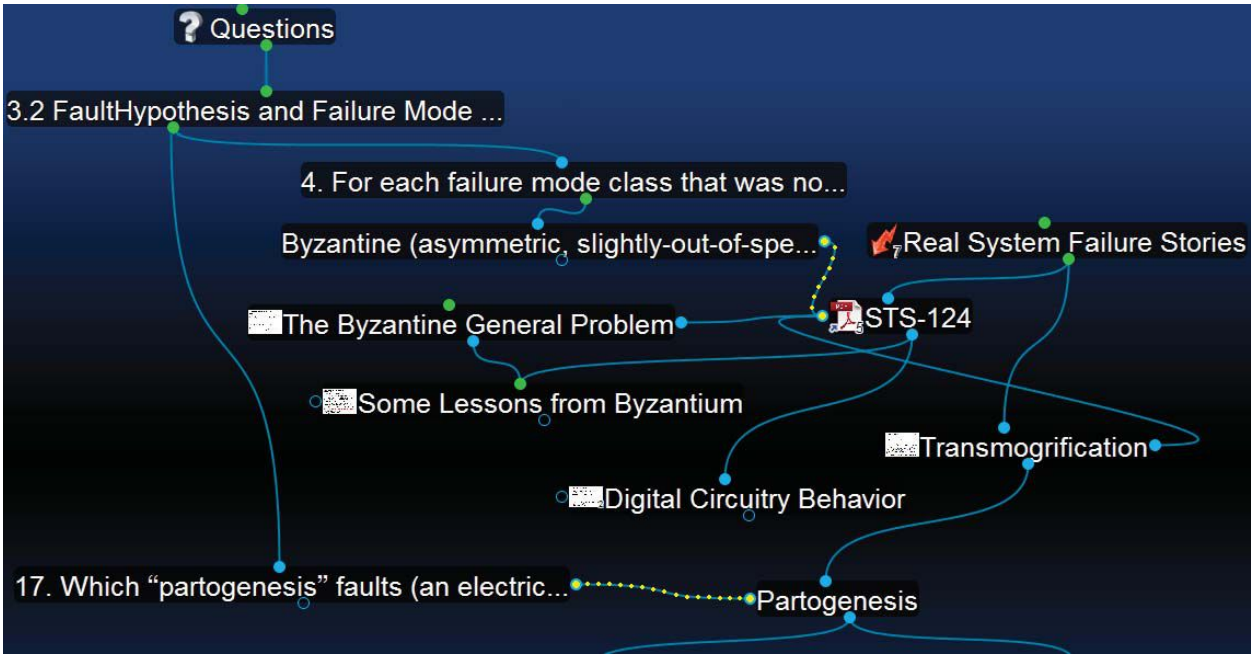

Figure A3. Linking Questions and Stories

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01- 07 - 2014	Contractor Report	September 2010 - October 2013

4. TITLE AND SUBTITLE

Distributed System Design Checklist

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Hall, Brendan; Driscoll, Kevin

5d. PROJECT NUMBER

5e. TASK NUMBER

NNL10AB32T

5f. WORK UNIT NUMBER

534723.02.02.07.30

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

NASA Langley Research Center
Hampton, Virginia 23681

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSOR/MONITOR'S ACRONYM(S)

NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

NASA/CR-2014-218504

12. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 62
Availability: NASA CASI (443) 757-5802

13. SUPPLEMENTARY NOTES

Langley Technical Monitor: Paul S. Miner

14. ABSTRACT

This report describes a design checklist targeted to fault-tolerant distributed electronic systems. Many of the questions and discussions in this checklist may be generally applicable to the development of any safety-critical system. However, the primary focus of this report covers the issues relating to distributed electronic system design. The questions that comprise this design checklist were created with the intent to stimulate system designers' thought processes in a way that hopefully helps them to establish a broader perspective from which they can assess the system's dependability and fault-tolerance mechanisms. While best effort was expended to make this checklist as comprehensive as possible, it is not (and cannot be) complete. Instead, we expect that this list of questions and the associated rationale for the questions will continue to evolve as lessons are learned and further knowledge is established. In this regard, it is our intent to post the questions of this checklist on a suitable public web-forum, such as the NASA DASHLink AFCS repository. From there, we hope that it can be updated, extended, and maintained after our initial research has been completed.

15. SUBJECT TERMS

Dependability; Distributed Systems; Fault-tolerance; Integrated modular avionics

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	42	(443) 757-5802